Süsteemprogrammeerimine keeles C

Üheksas loeng

Tänases loengus

- Ajainfo haldamine
- Sisend ja väljund üle socketite
- Ühenduse loomine
- Hosti ja Võrgu baidijärjekord
- bind(), listen(), accept()
- Saatmine ja vastuvõtt
- Ühenduse sulgemine
- Serveri näide

Üldine eesmärk

- Tekitada klient-server paar, mis võimdaldaks märkmete salvestamist
- Klient saadab sõnumi ja saab vastu 10 viimast salvestatud sõnumit ja nende ajad

Aeg

- Kuupäeva ja aja andmete muutmiseks vajalikud definitsioonid ja funktsiooniprototüübid leiab failist time.h .

```
typedef long time_t;     /* time value in seconds */
typedef long clock_t;    /* time value in clock ticks */
```

```
struct tm {
  int tm_sec;     /* seconds after the minute - [0,59] */
  int tm_min;     /* minutes after the hour - [0,59] */
  int tm_hour;    /* hours since midnight - [0,23] */
  int tm_mday;    /* day of the month - [1,31] */
  int tm_mon;     /* months since January - [0,11] */
  int tm_year;    /* years since 1900 */
  int tm_wday;    /* days since Sunday - [0,6] */
  int tm_yday;    /* days since January 1 - [0,365] */
  int tm_isdst;   /* daylight savings time flag */
};
```

clock()

- clock_t clock(void) tagastab programmi jooksutamiseks kulunud aja protsessori *tick*ides.
- Tegeliku aja saamiseks jaga clock() tulemus CLOCKS_PER_SEC väärtusega (POSIX: 1000000)
- Esimese väljakutse väärtus ei ole defineeritud: tegeliku aja saamiseks käivita programmi alguses clock ja lahuta tulemuseks saadud arv järgmiste väljakutsete tulemusest.
- Clock võib ringiga algusse tagasi jõuda (32 bitisel arhitektuuril 1000000 juures iga 72 tunni tagant)

time()

- time_t time(time_t *tptr) tagastab vastavalt süsteemi kellale kalendriaja (sekundite arv kesköösst (00:00) 1 jaanuaril 1970 (*Epoch*) UTC järgi).
- Kui tptr ei ole NULL, väärtustatakse pointeriga näidatud koht kalendriajaga.
- Implementatsioon on üsna naiivne, ei arvesta näiteks liigsekunditega (aatomikellades tehtavad korrektuurid vastavalt Maa pöörlemishäiretele)

struct tm, UTC, vööndiaeg

```
struct tm *gmtime(const time_t *timep);
struct tm *localtime(const time_t *timep);

extern char *tzname[2];
long int timezone;
extern int daylight;
```

- gmtime() lammutab sekundid komponentideks ja tagastab aja struktuuri UTC järgi
- localtime() lammutab samuti komponentideks, kuid annab kohaliku aja. Samuti väärtustab välised muutujad tzname ajatsooniga, timezone erinevusega UTC-st sekundites ning muutuja daylight, mis saab suveaja puhul nullist erineva väärtuse

Aeg stringiks

```
char *ctime(time_t *tptr);
```

- Teisendab sekunditega antud aja inimloetavale kujule:
  - "Tue Apr 16 10:49:08 1993\n"

```
char *asctime(const struct tm *tp);
```

- Teisendab struktuuriga antud aja inimloetavale kujule
- ctime(t) == asctime(localtime(c))

Ajavahemik

```
double difftime(time_t t0,  time_t t1);
```

- Tagastab kahe ajahetke vahe sekundites
  - Väärtus on t1 – t0
- Programmi tööaja arvutamiseks küsi kaks korda aeg, siis kutsu välja difftime()

struct tm -> sekundid

```
time_t mktime(struct tm *timeptr);
```

- Teisendab komponentideks muudetud aja kalendriaja sekunditeks.
  - Struktuuri liikmeid tm_wday ja tm_yday ignoreeritakse, ülejäänud numbritest arvutatakse uus aeg
  - Kui väärtused on väljaspool nende maksimaalväärtusi, aeg normaliseeritakse (40 okt = 9 nov). Samuti määratakse tzname.
  - Kui aeg ei ole kalendriajaks teisenduv (sekundid 1970a 1. jaanuarist), tagastatakse (time_t)(-1) ja tm_wday ja tm_yday liikmeid ei torgita.

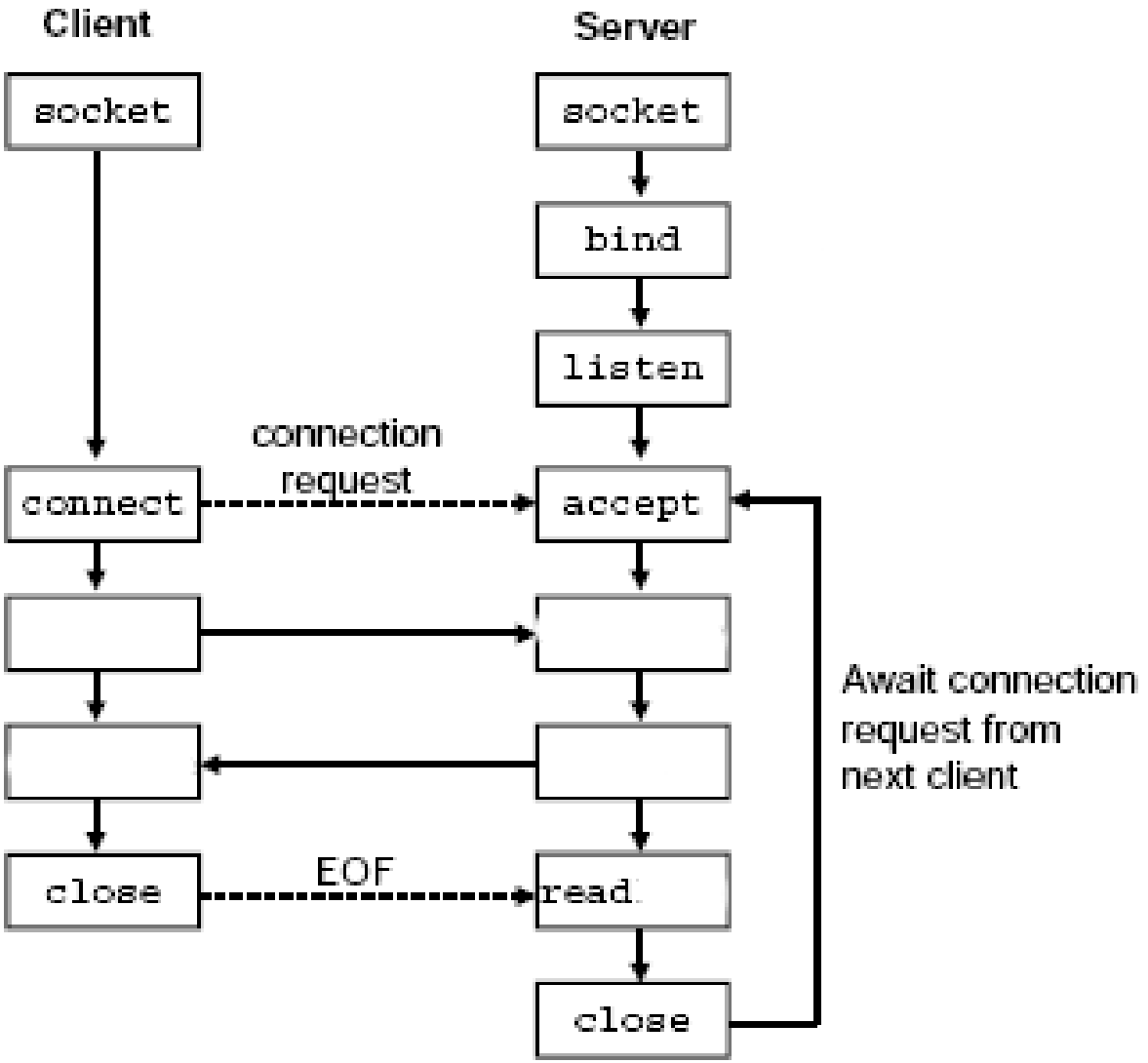
Võrgunduse alused

- Failideskriptor on avatud failiga seotud täisarv. See fail võib olla avatud võrguühendus, FIFO, terminal, tegelik kettal olev fail, või peaegu misiganes muu.
- **Unixis on kõik asjad failid!** Kui soovid üle interneti mõne teise programmiga suhelda, teed seda failideskriptori abil.

Socket

- Socket: viis suhelda teiste programmidega standardseid Unixi failideskriptoreid kasutades
- Socketeid on erinevat tüüpi, olulisemad:
  - Interneti socketid
  - Unixi socketid

Ülevaade socketi liidesest



Interneti socketid

- Interneti kasutamiseks on erinevaid socketi tüüpe, mis vastavad kõik mingi TCP/IP protokolli tasemele:
  - *Stream socket*: TCP protokoll
  - *Datagram socket*: UDP protokoll
  - *Raw socket*: Puhas IP pakett

Socketi tegemine

```
#include <sys/types.h>
#include <sys/socket.h>
int socket(int domain, int type, int protocol);
```

- domain: Protokollide perekond. PF_UNIX, PF_LOCAL Unixi kohalikud protokollid. PF_INET – Interneti perekond, PF_IPV6 – IP v6, PF_APPLETALK jne.
- type: SOCK_STREAM, SOCK_DGRAM, SOCK_RAW jne (levinumad kaks esimest)
- protocol (konkreetne protokoll (enamasti 0))
- tagastab "erilise" failideskriptori

Socket fd

- Socketi failideskriptor on veidi teiselaadne. Lisaks read() ja write() käskudele on seal ka spetsiaalsed võrgukäsud send() ja recv()
- Socketid, mille tüüp on SOCK_STREAM on kahepidise ühendusega.
- Stream tüüpi socket tuleb enne sinna kirjutamist või sealt lugemist kuhugi ühendada: connect()
- Pärast ühendumist on võimalik lugeda ja kirjutada nii send(), recv(), kui read() ja write() abil.
- Sessiooni lõppedes saab sulgeda close() abil.

Näidis

```
int sock
sock = socket(PF_INET, SOCK_STREAM,0);
if (sock == -1) return ERROR;
```

- Socketi fd on väike positiivne täisarv
- Väärtus -1 tähistab viga, samuti määratakse errno
- socket() on süsteemikäsk – TCP/IP on kernelisse sisseehitatud
- protocol 0 ütleb, et valik toimugu vastavalt perekonnale ja ühenduse tüübile.

Ühenduse loomine

```
int connect(int sockfd, struct sockaddr
  *serv_addr, socklen_t addrlen);
```

- sockfd – socket() poolt tagastatud deskriptor
- Kui socket on SOCK_DGRAM tüüpi, määratakse koht, kuhu ühendutakse
- Kui socket on SOCK_STREAM tüüpi, luuakse stream tüüpi ühendus
- Edu korral tagastub 0, ebaedu korral -1 ja errno
- Stream socketisse saab ühendada vaid korra, datagram socketisse saab uuesti ühenduda

Socketi aadressid

- Aadressi üldkuju:

```
struct sockaddr {
    unsigned short sa_family; // address family, AF_xxx
    char  sa_data[14];  // 14 bytes of protocol address
};
```

- sockaddr_in interneti jaoks:

```
truct sockaddr_in {
  short int          sin_family;  // Address family == AF_INET
  unsigned short int sin_port;    // Port number
  struct in_addr     sin_addr;    // Internet address
  unsigned char sin_zero[8]; // Padding to get correct size
};
```

  - sin_family on AF_INET ja sin_zero tuleb nt memset() abil ära nullida

Socketi aadress (2)

- Internetiaadress:

```
struct in_addr {
  unsigned long s_addr; // that's a 32-bit long, or 4 bytes
};
```

- Pordi ja võrguaadressi (aga mitte protokolli) baitide järjestus peab vastama võrgu baidijärjestusele (*network byte order*). Sellest järgmisel slaidil.

Hosti ja Võrgu baidijärjestus
Host and Network Byte Order

- Mitmebaidilist arvu on võimalik järjestada kaht pidi: kõige olulisem bait eespool ning kõige vähemolulisem bait eespool.
- Esimene neist ongi võrgu baidijärjestus
- Mõnes masinas on numbrid sisemiselt võrgu baidijärjestuses, mõnes (Intel) mitte.
- Konverteerimisfunktsioonid:

    htons() - host to network short

    htonl() - host to network long

    ntohs() - network to host short

    ntohl() - network to host long

Töö võrgunimede ja aadressidega

```
struct hostent *gethostbyname(const char *name);
struct hostent *gethostbyaddr(const char *addr, int
len, int type);
```

- gethostbyname() tagastab hostinimele vastava hostent struktuuri.
- gethostbyaddr() tagastab hostinimele, mille pikkus on len ja tüübile type vastava hostent struktuuri. Ainus lubatud type on AF_INET.
- Vea korral tagastavad mõlemad NULL

Töö võrgunimede ja aadressidega

```
struct hostent {
  char    *h_name;        /* official name of host */
  char    **h_aliases;    /* alias list */
  int     h_addrtype;     /* host address type */
  int     h_length;       /* length of address */
  char    **h_addr_list;  /* list of addresses */
}
#define h_addr  h_addr_list[0]  /* for backward compatibility */
```

h_name The official name of the host.

h_aliases A zero-terminated array of alternative names for the host.

h_addrtype The type of address; always AF_INET at present.

h_length The length of the address in bytes.

h_addr_list A zero-terminated array of network addresses for the host in network byte order. (no need to convert!!!!!!!)

h_addr The first address in h_addr_list for backward compatibility.

Näidis: postit klient

```
/*************************************************
   Simple example of network communication.

   The server maintains a collection of up to 10
   messages. Clients can connect to post a new
   message, and then recieve all the messages back.

   File postit.c

   Compile with:
   gcc -pedantic -Wall -lxnet postit.c -o postit
  *************************************************/

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <unistd.h>
#include <stdlib.h>
#include "postit.h"
```

```
int main(int argc, char *argv[]) {

  struct sockaddr_in server; /* Server's address assembled here */
  struct hostent *host_info;
  int sock, i, c;
  char line[MSG_LEN]; /* Buffer to copy from user to server */
  char *server_name;

  /* Get server name from commandline. If none, use localhost */
  if ( argc > 1 )
    server_name = argv[1];
  else
    server_name = "localhost";

  /* Create the socket */
  sock = socket(AF_INET, SOCK_STREAM, 0);
  if (sock < 0) {
    perror("creating stream socket");
    exit(EXIT_FAILURE);
  }

  host_info = gethostbyname(server_name);
  if (host_info == NULL) {
    fprintf(stderr, "%s: unknown host: %s\n", argv[0], server_name);
    exit(EXIT_FAILURE);
  }
```

postit.c (3)

```
/* Set up the server's socket address, then connect */
  server.sin_family = host_info->h_addrtype;
  memcpy((char *)&server.sin_addr, host_info->h_addr,
              host_info->h_length);
  server.sin_port = htons(POSTIT_TCP_PORT);
                // st. POSTIT_TCP_PORT==1066

  if (connect(sock, (struct sockaddr *)&server, sizeof(server)) < 0)
  {
    perror("connecting to server");
    exit(EXIT_FAILURE);
  }

  /* We are connected to the server.
     Read a line from the user, and write it to the socket.
     Read the return value from the server, and print it to the
     screen/standard output.
  */

  printf("connected to server %s\n", server_name);
```

postit.c (4)

```
for(i=0; i < MSG_LEN-1 && (c = getchar()) != EOF && c != '\n'; i++)
    line[i] = c;


  line[i] = '\0';

  write(sock, line, i+1);

  printf("\n\n");
  while ((i = read(sock, line, MSG_LEN-1)) > 0) {
    line[i] = '\0';
    printf("%s", line);
  }
  printf("\n\n");
  close(sock);

  return 0;
}
```

Serveri pool: bind()

```
int bind(int sockfd, struct sockaddr *my_addr, int
addrlen);
```

- bind() seob kohaliku socketi kohaliku aadressiga my_addr, mille pikkus on addrlen. Algselt on loodud socket küll seotud protokolliperekonnaga, aga sidumata mingite konkreetsete aadressidega.
- Enne kui SOCK_STREAM saab minna LISTEN seisundisse, et ühendusi oodata, on vaja bind abil kohalik aadress deskriptoriga siduda.
- -1 vea korral + errno
- Teise masinasse ühendumisel on bind kasutamata kohaliku pordiga automaatne.

bind() näide

```
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#define MYPORT 3490
main()
{
  int sockfd;
  struct sockaddr_in my_addr;

  sockfd = socket(AF_INET, SOCK_STREAM, 0);
      // do some error checking!

  my_addr.sin_family = AF_INET;         // host byte order
  my_addr.sin_port = htons(MYPORT);     // short, network byte
order
  my_addr.sin_addr.s_addr = inet_addr("10.12.110.57");
  memset(&(my_addr.sin_zero), '\0', 8); // zero the rest of the
struct
  // don't forget your error checking for bind():
  bind(sockfd, (struct sockaddr *)&my_addr, sizeof(struct
sockaddr));
```

Serveri pool: listen()

```
int listen(int s, int backlog);
```

- Ühenduste vastuvõtmiseks tehakse kõigepealt socket(), siis bind() ja seejärel antakse käsuga listen() teada soovist võtta vastu sissetulevaid ühendusi. Saabuvad ühendused võetakse vastu käsuga accept(). Listen kehtib vaid socketite puhul, mille tüüp on SOCK_STREAM või SOCK_SEQPACKET.
- backlog ütleb, mitut ühendust hoitakse järjekorras. Järjekorra täitumisel võib klient saada teate ECONNREFUSED või, kui protokoll lubab, ignoreeritakse seda edasise taas-saatmise huvides

Serveri pool: accept()

```
int accept(int s, struct sockaddr *addr, int *addrlen);
```

- s on socket, mis on socket() abil loodud, bind() abil seotud ja mis ootab pärast listen() käsku ühendusi.
  - accept() võtab järjekorrast esimese ühenduse, loob selle jaoks **uue** socketi samade parameetrite, kuid erineva deskriptoriga. Kui järjekord on tühi ja socket pole määratud *non-blocking* režiimi, blokeerub programmi töö kuni ühendus saabub. Kui ühendus on *non-blocking* ja järjekord on tühi, tagastub viga. Accept abil ei saa sama ühendust kaks korda vastu võtta, kuid esialgne socket jääb avatuks ja kuulama.

```
fcntl(sockfd, F_SETFL, O_NONBLOCK); // NON-BLOCKING
```

Näidis: listen(), accept()

```
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#define MYPORT 3490    // the port users will be connecting to
#define BACKLOG 10     // how many pending connections queue will hold
main()
{
  int sockfd, new_fd;  // listen on sock_fd, new connection on new_fd
  struct sockaddr_in my_addr;    // my address information
  struct sockaddr_in their_addr; // connector's address information
  int sin_size;
  sockfd = socket(AF_INET, SOCK_STREAM, 0); // do some error checking!
  my_addr.sin_family = AF_INET;         // host byte order
  my_addr.sin_port = htons(MYPORT);     // short, network byte order
  my_addr.sin_addr.s_addr = INADDR_ANY; // auto-fill with my IP
  memset(&(my_addr.sin_zero), '\0', 8); // zero the rest of the struct

  // don't forget your error checking for these calls:
  bind(sockfd, (struct sockaddr *)&my_addr, sizeof(struct sockaddr));
  listen(sockfd, BACKLOG);
  sin_size = sizeof(struct sockaddr_in);
  new_fd = accept(sockfd, (struct sockaddr *)&their_addr, &sin_size);
  ……. .
```

send(), recv()

```
int send(int sockfd, const void *msg, int len, int flags);
int recv(int sockfd, void *buf,
                                int len, unsigned int flags);
```

- send() tagastab tegelikult väljaläinud baitide arvu! Kasutaja peab seda kontrollima. Saatke väiksemaid faile.
- recv() tagastab failipuhvrist loetud baitide arvu või -1 vea korral (ja errno määratakse ka)
- Kui recv() tagastab 0, tähendab, et vastaspool sulges ühenduse
- Flag on enamasti 0, man-leht annab lisainfot

sendto() ja recvfrom()

```
int sendto(int sockfd, const void *msg, int len,
       int flags, const struct sockaddr *to, int tolen);
int recvfrom(int sockfd, void *buf, int len,
       unsigned int flags, struct sockaddr *from,
       int *fromlen);
```

- Sarnanevad send() ja recv()-ga, kuid lubavad ühendamata datagrammisocketitesse saatmiseks aadressi määrata.
- Kui ühendad datagrammisocketi connect() abil, võid ikka send() abil üle UDP infot saata, socketi tüüp ei muutu, puuduv aadress lisandub liideses endas.

Ühenduse sulgemine

```
int close(int sockfd);  // lihtsam viis
int shutdown(int sockfd, int how);
```

- Viimane pakub rohkem kontrolli:
  - 0 : edasine vastuvõtt keelatud
  - 1: edasine saatmine keelatud
  - 2: edasine saatmine ja vastuvõtt keelatud
- 0 edu korral -1 vea puhul (+ errno)
- shutdown() puhul on oluline märkida, et see ei sulge faili, vaid piirab kasutusvõimalusi. Socketi deskriptori vabastab ikka close()

Postit serveri näide

```
/*********************************************
   Simple example of network communication.

   The sever maintains a collection of up to 10
   messages. Clients can connect to post a new
   message, and then recieves all the messages back.

*********************************************/

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <stdio.h>
#include <errno.h>
#include <time.h>
#include <netdb.h>
#include <unistd.h>
#include <stdlib.h>
#include "postit.h"
```

```
#define NO_MSGS 10 /* Maximum number of messages on the board */

typedef struct _msg_data {
  char message[MSG_LEN];
  char time[30];
  int used;
} msg_data;

typedef struct _msg_board {
  msg_data entries[NO_MSGS];
  int next;
} msg_board;

/* Function prototypes: */
void get_message(int fd, struct sockaddr_in *addr, msg_board
*board);
void print_board(msg_board *board);
int mk_board(char *buf, msg_board *b);

int sock;
```

```
int main() {
  int client_fd, client_len, i;
  struct sockaddr_in server, client;
  msg_board board;
  char buf[NO_MSGS*(50+MSG_LEN)];

  /*    Initialize the message board */
  board.next = 0;
  for(i=0; i < NO_MSGS; i++){
    board.entries[i].used = 0;
  }

  /*    Start by creating a stream-socket  */
  sock = socket(AF_INET, SOCK_STREAM, 0);
  if (sock < 0) {
    perror("creating stream socket");
    exit(EXIT_FAILURE);
  }
```

```
/* Bind the socket to the predetermined port  */
  server.sin_family = AF_INET;
  server.sin_addr.s_addr = htonl(INADDR_ANY);
  server.sin_port = htons(POSTIT_TCP_PORT);

  if (bind(sock, (struct sockaddr *) &server,
                                     sizeof (server)) < 0) {
    perror("binding socket");
    exit(EXIT_FAILURE);
  }

  /*    Signal that we are ready to accept connections on sock
*/
  listen(sock, 5);
```

```
/*    Enter main loop  */
  while (1) {
    struct hostent *ht;

    client_len = sizeof(client);
    client_fd  = accept(sock, (struct sockaddr *)&client,
                                              &client_len);
    if (client_fd < 0) {
      perror("accepting connection");
      exit(EXIT_FAILURE);
    }

    ht = gethostbyaddr((char*) &client.sin_addr,
                       sizeof(client.sin_addr),
                       AF_INET);
    if (ht == NULL)
      printf("**gethostbyaddr failed**\n");
    else
      printf("**connection from <%s> **\n", ht->h_name);

    get_message(client_fd, &client, &board);
```

```
i = mk_board(buf, &board);
#ifdef  DEBUG
    printf("%s\n", buf);
#endif
    write(client_fd, buf, i);

    close(client_fd);

  } // end of while(1) loop

  return 0;
} //end of main()
```

```
/*      Get a message from fd and update the board.  */
void get_message(int fd, struct sockaddr_in *addr, msg_board
*board) {
  int n, next;
  time_t t;
  char *time_p;

  next = board->next;
  n = read(fd, board->entries[next].message, MSG_LEN);
  time(&t);
  time_p = ctime(&t);
  strcpy(board->entries[next].time, time_p);
  board->entries[next].used = 1;
  board->next = (next +1) % NO_MSGS;
}
```

```
/*  Compile board into printable string in buf  */
int mk_board(char *buf, msg_board *board){
  int i, n, len;
  len = 0;
  n = (board->next + NO_MSGS ) % NO_MSGS;
  for (i=0; i < NO_MSGS; i++, n = (n+1)%NO_MSGS) {
    len += sprintf(buf+len, "%2d:  ", NO_MSGS-i);
    if (board->entries[n].used != 0) {
      len += sprintf(buf+len, "%s", board->entries[n].time);
      len += sprintf(buf+len, "  %s\n",
                                  board->entries[n].message);
    }
    else
      len += sprintf(buf+len, "\n\n");
  }

  return len;
}

/********************* END OF EXAMPLE************************/
/* viimane slaid, olge _hästi vaikselt_ kuni näide lõppeb  */
```